ขจร โรจนเมธินทร์

SOFT MAIL

รู้จักสวิตช์บนเมนบอร์ด ด้วยโปรแกรม

โปรแกรมที่แสดงแบบจำลองของสวิตช์ที่อยู่บนเมนบอร์ด

ปัจจุบันเป็นยุคที่มีการพัฒนาทางคอมพิวเตอร์ไปอย่างรวดเร็วมาก ถ้าลองย้อนหลังไปประมาณ 6-7 ปี เครื่องคอมพิวเตอร์รุ่นพีซีและเอกซ์ทีดูจะเป็นเครื่องที่ทันสมัยมาก เพราะเพิ่งออกสู่ตลาดไม่นาน ปัจจุบันนี้เครื่องรุ่นดังกล่าวดูล้าสมัยไปแต่คิดว่ายังมีผู้ใช้อยู่อีกกลุ่มหนึ่งที่ยังใช้เครื่องรุ่นเอกซ์ที (รวมทั้งผู้เขียนด้วย) บทความเรื่องนี้จึงขออนุญาตเขียนสิ่งที่ค่อนข้างเกี่ยวกับเครื่องคอมพิวเตอร์รุ่นพีซีและเอกซ์ที แต่สำหรับผู้อ่านที่ใช้เครื่องรุ่นเอทีก็จะได้รับประโยชน์จากบทความนี้บ้าง

แนะนำตัวกันก่อน

XTSYS เป็นโปรแกรมที่เขียนด้วย Turbo Pascal โดยการแสดงผลบนจอภาพจะทำในโหมดกราฟิก ดังนั้นสิ่งที่จำเป็นก่อนที่จะรันโปรแกรมคือจะต้องก๊อบปี้ไฟล์นามสกุล .BGI และ CHR ซึ่งเป็นไฟล์ไดรฟ์เวอร์และไฟล์ตัวอักษรมาอยู่ในไดเรกทอรีเดียวกับโปรแกรม XTSYS เนื่องจาก XTSYS จะต้องโหลดไฟล์เหล่านี้ลงบนหน่วยความจำก่อน ยกเว้นแต่จะลิงค์ไฟล์เหล่านี้ไว้กับ XTSYS เลย

หลังจากที่รัน XTSYS แล้ว โปรแกรมจะแสดงผลการทำงานทางจอภาพตามรูปที่ 1 ซึ่งจะสรุปการทำงานได้ดังนี้

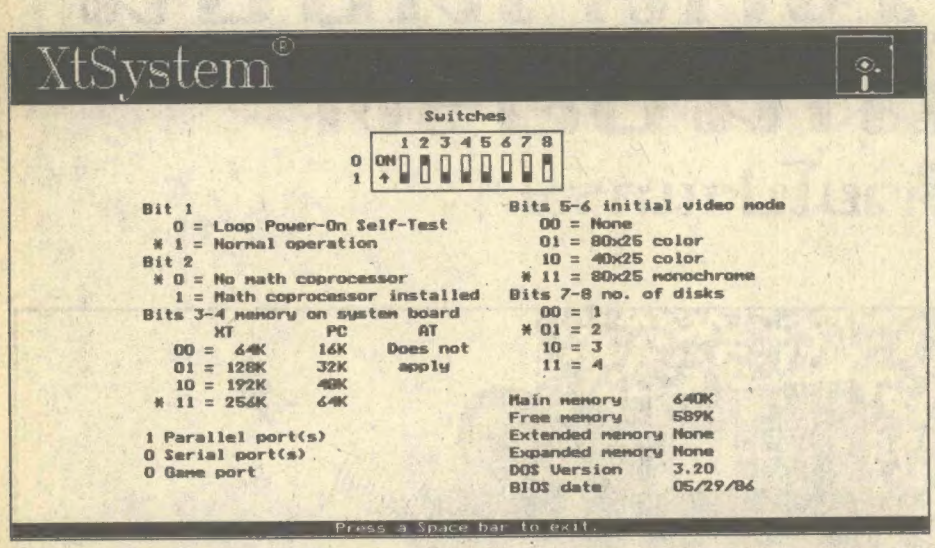
1. แสดงแบบจำลองของสวิตช์ที่อยู่บนเมนบอร์ด และลักษณะการเปิด-ปิดของสวิตช์คือ ถ้าสี่เหลี่ยมเล็ก ๆ นั้นอยู่ในตำแหน่ง 0 (บน) แสดงว่าสวิตช์เปิด แต่ถ้าอยู่ตำแหน่ง 1 (ล่าง) แสดงว่าสวิตช์ปิด หมายเลข 1-8 ใช้อ้างอิงถึงบิต 0-7 ของไบต์ต่ำของข้อมูลขนาด 1 เวิร์ด ที่แอดเดรส 40:10h ในไบออสซึ่งข้อมูลดังกล่าวจะเก็บรายละเอียดที่เกี่ยวกับเครื่องคอมพิวเตอร์ไว้

2. แสดงจำนวนพอร์ตแต่ละชนิดที่มีในเครื่องคอมพิวเตอร์

3. แสดงขนาดของหน่วยความจำหลักบนเมนบอร์ดและหน่วยความจำที่ยังเป็นอิสระอยู่

4. แสดงขนาดของหน่วยความจำแบบ Extended และ Expanded

5. แสดงเวอร์ชันของดอสที่ใช้งาน และวันที่ของไบออสของเครื่อง



รูปที่ 1 แสดงจอภาพหลังจากการรัน XTSYS

มีข้อมูลอะไรเก็บที่แอดเดรส 40:10h

เมื่อเราเปิดเครื่องคอมพิวเตอร์ โปรแกรมไบออส ส่วน POST (Power–On Self-Test) ในเครื่องรุ่นพีซีหรือเอกซ์ทีจะตรวจสอบสถานะการเปิดปิดของสวิตช์หมายเลข 1-8 (ดูรายละเอียดที่ตารางที่ 1) ที่ถูกตั้งไว้ก่อนแล้ว สำหรับเครื่องรุ่นเอทีข้อมูลเหล่านี้จะถูกตั้งด้วยการรันโปรแกรม Diagnostic setup

รูทีน BIOS POST จะอ่านสถานะการเปิดปิดของสวิตช์ของพอร์ตหมายเลข 61h และ 62h พร้อมทั้งจะตรวจสอบจำนวนการติดตั้งพอร์ตอนุกรม, พอร์ตขนาน, พอร์ตเกม จากนั้นจะนำข้อมูลเหล่านี้ไปเก็บไว้ที่แอดเดรส 40:10h ในไบออส ซึ่งผู้ใช้สามารถจะอ่านเอาข้อมูลนั้นมาใช้ได้ผ่านทางอินเตอร์รัพต์ 11h โดยจะได้ข้อมูลคืนมาทางรีจิสเตอร์ AX ซึ่งค่าในไบต์ต่ำ (รีจิสเตอร์ AL) จะแสดงข้อมูลตามตารางที่ 1 และค่าในไบต์สูง (รีจิสเตอร์ AH) จะแสดงข้อมูลจำนวนพอร์ตต่าง ๆ ที่ติดตั้งอยู่ตามตารางที่ 2

จากตารางที่ 1 การแปลหน้าที่ของตำแหน่งสวิตช์ทั้ง 8 นั้นใช้ตามหนังสือคู่มืออ้างอิงทางเทคนิคของเครื่องเอกซ์ทีทั้งนี้เนื่องจากในเครื่องคอมแพตทิเบิ้ลบางยี่ห้อนั้นแปลต่างจากของเครื่องไอบีเอ็ม เช่น ตำแหน่ง 3-4 ที่แทนขนาดหน่วยความจำ บางเครื่องอาจไม่ใช้ เนื่องจากเครื่องรุ่นเอกซ์ทีเดิมมีขนาดหน่วยความจำบนเมนบอร์ดได้มากที่สุด 256K แต่ขยายได้ถึง 640K โดยใช้การ์ดเสียบ หรือในเครื่องไอบีเอ็มรุ่นเอทีก็ไม่ได้ใช้บิตที่ 2-3 ของไบต์ต่ำที่แอดเดรส 40:10h เพื่อแทนขนาดหน่วยความจำเป็นต้น

การตรวจสอบค่าของแต่ละบิตที่ได้มาจากอินเตอร์รัพต์ 11h จะทำโดยกำหนด MASK เท่ากับ 80h (10000000b) แล้วนำมา and กับค่าที่เราต้องการตรวจสอบ ถ้าได้ผลลัพธ์เป็นศูนย์แสดงว่าบิตที่ 7 ของค่านั้นเป็นศูนย์ จากนั้นจะทำ shr และ and จนถึงบิตที่ 0 เราก็จะทราบค่าของแต่ละบิต

ตำแหน่ง	หน้าที่
1	แสดงสถานะการทำงานปกติ
2	แสดงการติดตั้งโคโปรเซสเซอร์
3-4	แสดงขนาดของหน่วยความจำบนเมนบอร์ด
5-6	แสดงชนิดของอะแดปเตอร์ของจอภาพ
7-8	แสดงจำนวนดิสก์ไดรฟ์ขนาด 5.25 นิ้ว

ตารางที่ 1 แสดงหน้าที่ของสวิตช์ทั้ง 8 ตำแหน่ง

บิตที่	หน้าที่
15-14	จำนวนเครื่องพิมพ์ที่ติดตั้ง
13	ไม่ใช้
12	แสดงการติดตั้งพอร์ตเกม
11-9	จำนวนพอร์ตอนุกรม (RS232)
8	ไม่ใช้

ตารางที่ 2 แสดงหน้าที่ของบิตที่ 8-15 ของเวิร์ดที่แอดเดรส 40:10h

อินเตอร์รัพต์ 12h ช่วยงานอะไรบ้าง

หน้าที่ของ POST อีกอย่างหนึ่งคือจะตรวจสอบขนาดหน่วยความจำหลักและเก็บค่าไว้ที่แอดเดรส 40:13h ในไบออสขนาด 1 เวิร์ด ซึ่งสามารถเรียกมาใช้ได้ทางอินเตอร์รัพต์ 12h โดยจะได้ขนาดของหน่วยความจำในหน่วยกิโลไบต์คืนมาทางรีจิสเตอร์ AX

สำหรับการหาขนาดของหน่วยความจำอิสระทำได้โดยหลังจากได้รัน XTSYS แล้วดอสจะโหลด XTSYS เป็นโปรแกรมสุดท้ายในหน่วยความจำ ดังนั้นขนาดของหน่วยความจำที่เป็นอิสระคือขนาดบล็อกของหน่วยความจำที่ XTSYS ใช้รวมกับขนาดบล็อกที่อยู่เหนือส่วนที่ XTSYS ใช้บวกกับขนาดบล็อกของหน่วยความจำของเอนวิรอนเมนต์ของ XTSYS

การหาขนาดบล็อกของหน่วยความจำที่ XTSYS ใช้นั้นทำได้โดยเราต้องค้นหา Memory Control Block (MCB) ที่ควบคุมหน่วยความจำส่วนนี้ ซึ่งจะอยู่ก่อน Program Segment Prefix (PSP) หนึ่งพารากราฟ ดังนั้นจึงต้องอ่านค่าเซกเมนต์ของ PSP จากตัวแปร PrefixSeg มาลบด้วยหนึ่ง ก็จะได้เป็นเซกเมนต์ของ MCB ซึ่งขนาดบล็อกของหน่วยความจำที่ MCB คุมจะเก็บอยู่ที่ออฟเซ็ตที่ 3 จากนั้นจะค้นหา MCB ที่อยู่ถัดไปเพื่อนำขนาดบล็อกมาบวกกับขนาดบล็อกของ XTSYS การค้นหา MCB จะหยุดเมื่อถึง MCB อันสุดท้าย สำหรับการค้นหา MCB ถัดไปนั้นเราจะทราบเซกเมนต์ของ MCB ถัดไปโดยนำค่าที่ออฟเซ็ตที่ 1 ซึ่งเป็นเซกเมนต์ของบล็อกที่ MCB อันปัจจุบันคุมอยู่บวกกับขนาดของบล็อกและถ้าค่าในออฟเซ็ตที่ 0 ของ MCB ถัดไปเท่ากับ 5Ah ก็แสดงว่าเป็น MCB อันสุดท้าย ถ้าไม่ใช่ค่าในออฟเซ็ตที่ 0 จะเป็น 4Dh

การหาขนาดบล็อกของเอนวิรอนเมนต์ของ XTSYS นั้นเราจะอ่านค่าเซกเมนต์ของเอนวิรอนเมนต์จากออฟเซ็ต 2Ch ในส่วน PSP มาลบหนึ่งก็จะได้เซกเมนต์ของ MCB ที่คุมบล็อกของเอนวิรอนเมนต์ขนาดของบล็อกจะอยู่ที่ออฟเซ็ตที่ 3 เช่นกัน สำหรับขนาดของหน่วยความจำอิสระที่หาได้จะมีหน่วยเป็นพารากราฟ (1 พารากราฟเท่ากับ 16 ไบต์) ดังนั้นจะเปลี่ยนเป็นกิโลไบต์โดยนำมาหาร 1024 แล้วคูณ 16

หาขนาดหน่วยความจำแบบ Extended

การติดตั้งหน่วยความจำแบบ Extended นั้นสามารถใช้ได้กับเครื่องรุ่นเอทีขึ้นไป ดังนั้นก่อนที่จะหาขนาดของหน่วยความจำเราจะต้องตรวจสอบรุ่นของเครื่องคอมพิวเตอร์ก่อน ทำได้โดยตรวจค่าที่แอดเดรส F000:FFFFh ขนาด 1 ไบต์ในไบออส (ดูค่าที่ตารางที่ 3) หลังจากที่ตรวจสอบว่าเป็นเครื่องรุ่นเอทีแล้ว เราจะเรียกอินเตอร์รัพต์ 15h ฟังก์ชัน 88h ซึ่งจะคืนค่าขนาดของหน่วยความจำแบบ Extended ในหน่วยกิโลไบต์มาทางรีจิสเตอร์ AX

ค่า	รุ่นเครื่อง
FF	พีซีหรือก่อนรุ่นเอกซ์ที
FE	เอกซ์ที
FD	เจอาร์
FC	เอที

ตารางที่ 3 แสดงความหมายของไบต์ที่แอดเดรส F000:FFFFh

หาขนาดหน่วยความจำแบบ Expanded

การใช้งานหน่วยความจำที่เป็น Expanded นั้นจะต้องมีการติดตั้งดีไวซ์ไดรฟ์เวอร์ ซึ่งจะเรียกผ่านทางอินเตอร์รัพต์ 67h การตรวจสอบว่ามีการติดตั้งดีไวซ์ไดรฟ์เวอร์อยู่หรือไม่ทำได้โดยใช้อินเตอร์รัพต์ 21h ฟังก์ชัน 35h อ่านค่าแอดเดรสของอินเตอร์รัพต์ 67h มาจากนั้นก็จะตรวจสอบในส่วนเฮดเดอร์ที่ออฟเซ็ต 0Ah ว่ามีชื่อดีไวซ์ "EMMXXXX0" อยู่หรือไม่ถ้ามีก็แสดงว่ามีดีไวซ์ไดรซ์เวอร์ติดตั้งอยู่หลังจากนั้นจะใช้ฟังก์ชัน 42h ของอินเตอร์รัพต์ 67h อ่านขนาดหน่วยความจำแบบ Expanded คืนมาทางรีจิสเตอร์ DX เป็นจำนวนบล็อกขนาด 16 กิโลไบต์

การหาเวอร์ชันของดอสและวันที่ของไบออส

การหาเวอร์ชันของดอสทำได้โดยการเรียกฟังก์ชัน 30h ของดอส ซึ่งดอสจะคืนเลขเวอร์ชันหลักแรกมาทางรีจิสเตอร์ AL และเลขเวอร์ชันหลังจุดทศนิยมมาทางรีจิสเตอร์ AH

วันที่ซึ่งแสดงรุ่นของไบออสนั้นโดยส่วนมากแล้วผู้ที่ออกแบบไบออสส่วนใหญ่จะออกแบบให้คล้ายกับของไอบีเอ็ม ซึ่งในไบออสของไอบีเอ็มจะเก็บวันที่ของไบออสไว้ที่แอดเดรส F000:FFF5h ดังนั้นถ้าไบออสของเครื่องคอมพิวเตอร์ที่คุณใช้ไม่เก็บไว้ที่แอดเดรสนี้ XTSYS ก็จะไม่แสดงว่าวันที่ของไบออสออกมา

ก่อนจะจบ

ถ้าท่านอ่านจากตอนต้นมาถึงตรงนี้ คิดว่าผู้อ่านทุกท่านทั้งผู้ที่ใช้เครื่องรุ่นพีซี เอกซ์ที หรือเอที ก็คงได้รับประโยชน์เล็กน้อยจากบทความนี้ และอย่างน้อย XTSYS ก็คงช่วยให้เราไม่ต้องเปิดฝาเครื่องถ้าต้องการทราบว่าสวิตช์บนเมนบอร์ดนั้นตั้งเอาไว้เท่าไร

โปรแกรมนี้ผู้เขียนให้เตรียมทำเป็นซอฟต์เมล์ไว้ให้เรียบร้อยแล้ว สำหรับผู้อ่านที่ไม่อยากคีย์โปรแกรมยาวเหยียด สามารถติดต่อไปยังกองบรรณาธิการวารสารไมโครเพื่อขอใช้บริการนี้ได้เช่นเคย

โปรแกรม XTSYS

```
program xtswitches;

uses
  crt, dos, graph;

var
  error_code   : integer;
  max_x, max_y : word;
  cur_x, cur_y : word;
  bit  : array [0..7] of byte;
  regs : registers;
  exitsave : pointer;

{$F+}
procedure myexit;
begin
  closegraph;
  if exitcode = 7 then
  begin
    delay(500);
    writeln('Memory Control Block Error')
  end;
  exitproc := exitsave
end;
{$F-}

function graph_setup : integer;
  var
    gr_driver, gr_mode, code : integer;
begin
  exitsave := exitproc;
  exitproc := @myexit;
  gr_driver := detect;
  initgraph(gr_driver,gr_mode,'');
  code := graphresult;
  if code = grok then
  begin
    max_x := getmaxx;
    max_y := getmaxy;
    cur_x := 0;
    cur_y := 0
  end;
  graph_setup := code
end;

procedure gr_write(s : string);
begin
  outtextxy(cur_x,cur_y,s);
```

```
    inc(cur_x,textwidth(s));
    if cur_x >= max_x then
    begin
      cur_x := 0;
      inc(cur_y,textheight(s)+4)
    end
  end;

  procedure gr_write_int(x : word);
    var
      s : string;
  begin
    str(x,s);
    gr_write(s)
  end;

  procedure gr_write_bk(s : string);
  begin
    setcolor(0);
    gr_write(s);
    setcolor(1)
  end;

  procedure gomyxy(x, y : word);
  begin
    cur_x := x; cur_y := y
  end;

  procedure show_diskette;
    const
      myfillpatt1 : fillpatterntype = (0,
      0,0,0,0,0,0,0);
      myfillpatt2 : fillpatterntype = ($FF,
      $FF,$FF,$FF,$FF,$FF,$FF,$FF);
  begin
    rectangle(max_x-70,6,max_x-26,46);
    circle(max_x-48,26,3);
    setfillpattern(myfillpatt1,0);
    floodfill(max_x-48,26,0);
    circle(max_x-48,26,7);
    arc(max_x-48,35,0,180,2);
    line(max_x-50,35,max_x-50,43);
    line(max_x-46,35,max_x-46,43);
    arc(max_x-48,43,180,360,2);
    floodfill(max_x-48,39,0);
    circle(max_x-38,30,2);
    floodfill(max_x-38,30,0);
    rectangle(max_x-28,13,max_x-26,15);
    floodfill(max_x-27,14,0);
    arc(max_x-54,46,0,180,2);
    floodfill(max_x-54,46,0);
    arc(max_x-42,46,0,180,2);
    floodfill(max_x-42,46,0);
    setfillpattern(myfillpatt2,1)
  end;

  procedure show_banner;
    var
      textinfo : textsettingstype;
  begin
    rectangle(0,0,max_x,max_y);
    gettextsettings(textinfo);
    settextstyle(triplexfont,horizdir,5);
    bar(2,2,max_x-2,textheight('K')+12);
    setcolor(0);
    gomyxy(2,1);
    gr_write(' XtSystem');
    settextstyle(smallfont,horizdir,4);
    gomyxy(cur_x-2,cur_y+4);
    gr_write(' R');
    circle(cur_x-4,cur_y+5,8);
    with textinfo do
      settextstyle(font,direction,
      charsize);
    show_diskette;
    setcolor(1)
  end;

  procedure show_switch;
    var
      s : string;
      x : word;
  begin
    s := 'Switches';
    gomyxy((max_x - textwidth(S)) div
    2,56);
    gr_write(s);
    x := (max_x-150) div 2;
    rectangle(x,70,x+150,110);
    s := '  1 2 3 4 5 6 7 8';
    gomyxy((max_x - textwidth(s)) div
    2,74);
    gr_write(s);
    gomyxy(X-15,86);
    gr_write('0');
    gomyxy(x-15,98);
    gr_write('1');
    gomyxy(x+7,98);
    gr_write(#24);
    gomyxy(x+4,86);
    gr_write('ON')
  end;

  procedure checkbit(x : byte);
    var
      i, mask : byte;
  begin
    mask := $80;
    for i := 7 downto 0 do
    begin
      if (x and mask) = 0 then
        bit[i] := 0
      else
        bit[i] := 1;
      mask := mask shr 1
    end
  end;

  procedure showbit;
    var
      i : byte;
      x : word;
  begin
    x := (max_x-150) div 2 + 23;
    for i := 0 to 7 do
      if bit[i] = 0 then
      begin
        bar(x+i*16,86,x+i*16+6,92);
        rectangle(x+i*16,92,x+i*16+6,104)
      end
      else
      begin
        rectangle(x+i*16,86,x+i*16+6,98);
        bar(x+i*16,98,x+i*16+6,104)
      end
  end;

  procedure markdetail(i, j : byte; x, y :
  word);
    const
```

```
    bin : array [0..1,0..1] of byte =
    ((0,1),(2,3));
begin
  gomyxy(x,y+bin[i,j]*(textheight('K')+
  4));
  gr_write('*')
end;

procedure showdetail;
  const
    texts : array [1..12] of string = (
    'Bit 1                                  Bits 5-6 initial video mode',
    '   0 = Loop Power-On Self-Test            00 = None',
    '   1 = Normal operation                   01 = 80x25 color',
    'Bit 2                                     10 = 40x25 color',
    '   0 = No math coprocessor                11 = 80x25 monochrome',
    '   1 = Math coprocessor installed      Bits 7-8 no. of disks',
    'Bits 3-4 memory on system board           00 = 1',
    '       XT          PC        AT           01 = 2',
    '   00 =  64K      16K     Does not        10 = 3',
    '   01 = 128K      32K      apply          11 = 4',
    '   10 = 192K      48K',
    '   11 = 256K      64K');

  var
    i : byte;
begin
  for i := 1 to 12 do
  begin
    gomyxy(105,118+(textheight('K')+4)*
    (i-1));
    gr_write(texts[i])
  end;
  markdetail(0,bit[0],113,130);
  markdetail(0,bit[1],113,166);
  markdetail(bit[3],bit[2],113,214);
  markdetail(bit[5],bit[4],402,130);
  markdetail(bit[7],bit[6],402,190)
end;

procedure show_parallel(x : byte);
  const
    mask : byte = $80;
    i : byte = 0;
begin
  if (x and mask) <> 0 then inc(i,2);
  mask := mask shr 1;
  if (x and mask) <> 0 then inc(i);
  gomyxy(105,274);
  gr_write_int(i);
  gr_write(' Parallel port(s)')
end;

procedure show_serial(x : byte);
  const
    mask : byte = $4;
    i : byte = 0;
begin
  if (x and mask) <> 0 then inc(i,2);
  mask := mask shr 1;
  if (x and mask) <> 0 then inc(i);
  gomyxy(105,286);
  gr_write_int(i);
  gr_write(' Serial port(s)')
end;

procedure show_game(x : byte);
  const
    mask : byte = $10;
    i : byte = 0;
begin
  if (x and mask) <> 0 then inc(i);
  gomyxy(105,298);
  gr_write_int(i);
  gr_write(' Game port')
end;

procedure show_equipment;
begin
  show_switch;
  intr($11,regs);
  checkbit(regs.al);
  showbit;
  showdetail;
  show_parallel(regs.ah);
  show_serial(regs.ah);
  show_game(regs.ah)
end;

procedure show_memory;
begin
  gomyxy(392,250);
  gr_write('Main memory      ');
  intr($12,regs);
  gr_write_int(regs.ax);
  gr_write('K')
end;

procedure show_free;
  var
    prgsize, envsize, envseg, fseg,
     blocksize : word;
    total : longint;
begin
  gomyxy(392,262);
  gr_write('Free memory      ');
  envseg := memw[prefixseg:$2C];
  envsize := memw[envseg-1:3];
  fseg := prefixseg-1;
  prgsize := 0;
  while mem[fseg:0] = $4D do
  begin
    blocksize := memw[fseg:3];
    inc(prgsize,blocksize);
    fseg := memw[fseg:1] + memw[fseg:3]
```

```
    end;
    if mem[fseg:0] = $5A then
    begin
      blocksize := memw[fseg:3];
      inc(prgsize,blocksize);
      inc(prgsize,envsize);
      total := round( (prgsize/1024) * 16 );
      gr_write_int(total);
      gr_write('K')
    end
    else
      halt(7)
 end;

 procedure show_extend;
 begin
   gomyxy(392,274);
   gr_write('Extended memory ');
   if mem[$F000:$FFFE] = $FC then
   begin
     regs.ah := $88;
     intr($15,regs);
     if regs.ax = 0 then
        gr_write('None')
     else
     begin
       gr_write_int(regs.ax);
       gr_write('K')
     end
   end
   else
     gr_write('None')
 end;

 procedure show_expand;
   var
     ts : string[8];
     i  : byte;
 begin
   gomyxy(392,286);
   gr_write('Expanded memory ');
   regs.al := $67;
   regs.ah := $35;
   msdos(regs);
   ts := '';
   for i := 1 to 8 do
     ts := concat(ts,char(mem[regs.es:
     $000A+(i-1)]));
   if ts = 'EMMXXXX0' then
   begin
     regs.ah := $42;
     intr($67,regs);
     gr_write_int(regs.dx*16);
     gr_write('K')
   end
   else
     gr_write('None')
 end;

 procedure show_dosver;
 begin
   gomyxy(392,298);
   gr_write('DOS Version      ');
   regs.ah := $30;
   msdos(regs);
   gr_write_int(regs.al);
   gr_write('.');
   gr_write_int(regs.ah)
 end;

 procedure show_bios_date;
   var
     i : byte;
     c : char;
 begin
   gomyxy(392,310);
   gr_write('BIOS date        ');
   i := 0;
   repeat
     c := char(mem[$F000:$FFF5+i]);
     gr_write(c);
     inc(i)
   until not (c in [' '..'z']);
   if (i >= 1) and (i <= 6) then
   begin
     gomyxy(520,310);
     gr_Write('Not found')
   end
 end;

 procedure status;
 begin
   setvisualpage(0);
   setactivepage(1);
   cleardevice;
   show_banner;
   show_equipment;
   show_memory;
   show_free;
   show_extend;
   show_expand;
   show_dosver;
   show_bios_date;
   setvisualpage(1)
 end;

 procedure waitkey;
   var
     c : char;
     s : string;
 begin
   s := 'Press a Space bar to exit.';
   bar(2,max_y-11,max_x-2,textheight
   ('K')+337);
   gomyxy((max_x - textwidth(s)) div 2,
   max_y-10);
   gr_write_bk(s);
   C := #00;
   repeat
     if keypressed then c := readkey
   until C = #32
 end;

 begin
   checkbreak := false;
   error_code := graph_setup;
   if error_code = grok then
   begin
     status;
     waitkey;
     closegraph;
     delay(500);
     writeln('XtSystem, Version 1.00, (C)
      Copyright. 1991, KJ')
   end
   else
     writeln('Graphic Error : ',grapherrormsg
     (error_code))
 end.
```